Assignment 4-K means

Siwei Li

October 30, 2015

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

Before starting this part, we first load two packages: fpc and flexclust. The package fpc will allow us to use the command plotcluster() to visualize the clustering result. The package flexclust will allow us to use the function randIndex() to evaluate the clustering result.

```
library('fpc')
library('flexclust')
```

```
## Loading required package: grid
## Loading required package: lattice
## Loading required package: modeltools
## Loading required package: stats4
```

Instead of using the built-in function kmeans() directly, we try to write the K means algorithm by ourselves. The function is called clusterByKmeans() with two parameter inputs: the data set and the number of groups.

```
clusterByKmeans <- function(dataset, groupNo){

# Obtain the number of rows and the number of columns of the dataset
rowNo <- nrow(dataset)
colNo <- ncol(dataset)

# Create a vector g to store the clustering result
# (indicating which point belongs to which group)
cluster <- vector("numeric", length=rowNo)

# Create a vector to store the distance between each data point
# and the center of the cluster to which that data point belongs.
dtc <- vector("numeric", length=rowNo)

# Create a matrix to store the centers of the groups
center <- matrix(0,nrow=groupNo,ncol=colNo)

# Randomly select 3 points (rows of data.train) as the initial centers of the 3 groups
randomInt <- as.vector(sample(1:rowNo,size=groupNo))
for (i in 1:groupNo){
    center[i,] <- dataset[randomInt[i],]
    center <- matrix(center,groupNo,colNo)
}

# We specify an intial way of clustering:
# all the rest of the points belong to the first cluster,
# while the points chosen as the initial centers belong to its own cluster.
for (i in 1:rowNo){
    cluster[i] <- 1
```

```
}
for (j in 1:groupNo){
    cluster[randomInt[j]] <- j
}


# Calculate the initial value of the distance vector, dtc
for (i in 1:rowNo){
    dtc[i] <- sqrt(sum((dataset[i,]-center[cluster[i],])^2))
}

# Create a variable, changed, which indicates
# whether the clustering result is changed in an iteration.
changed <- TRUE

# If the assignment is changed in a iteration, we continue;
# if the assignment no longer changes, we stop.
while(changed){

initialCluster <- cluster

for (i in 1:rowNo){
    initialdtc <- dtc[i]
    preCluster <- cluster[i]

    # We shall find a better cluster solution for the point i.
    for (j in 1:groupNo){
        # Compute the distance between the point i and the center of the cluster j
        currentdtc <- sqrt(sum((dataset[i,]-center[j,])^2))
        if (currentdtc<initialdtc){
            # Update the clustering result
            cluster[i] <- j
            # Update the distance vector
            dtc[i] <- currentdtc
        }
    }
  }

# Compare the new assignment with the previous assignment.
# The overall assignment is considered as "changed" if the assignment of
# at least one point is changed.
for (i in 1:rowNo){
    if (cluster[i] != initialCluster[i]){
        changed <- TRUE
        break
    }else{
        changed <- FALSE
    }
}

# Then we update the centers of each cluster under the new assignment.
for (k in 1:groupNo){
        # Obtain all the points in the cluster k
```

```
        pointsIn <- dataset[cluster==k,]
        pointsMat <- as.matrix(pointsIn)
        # If there is at least one points in the cluster k, we update the center
        # of the cluster k; otherwise, the center remains unchanged.
        if (nrow(pointsMat)>0){
            center[k,] <- colMeans(pointsMat)
        }else{
            center[k,] <- center[k,]
        }
      }
 }
return(cluster)
}
```

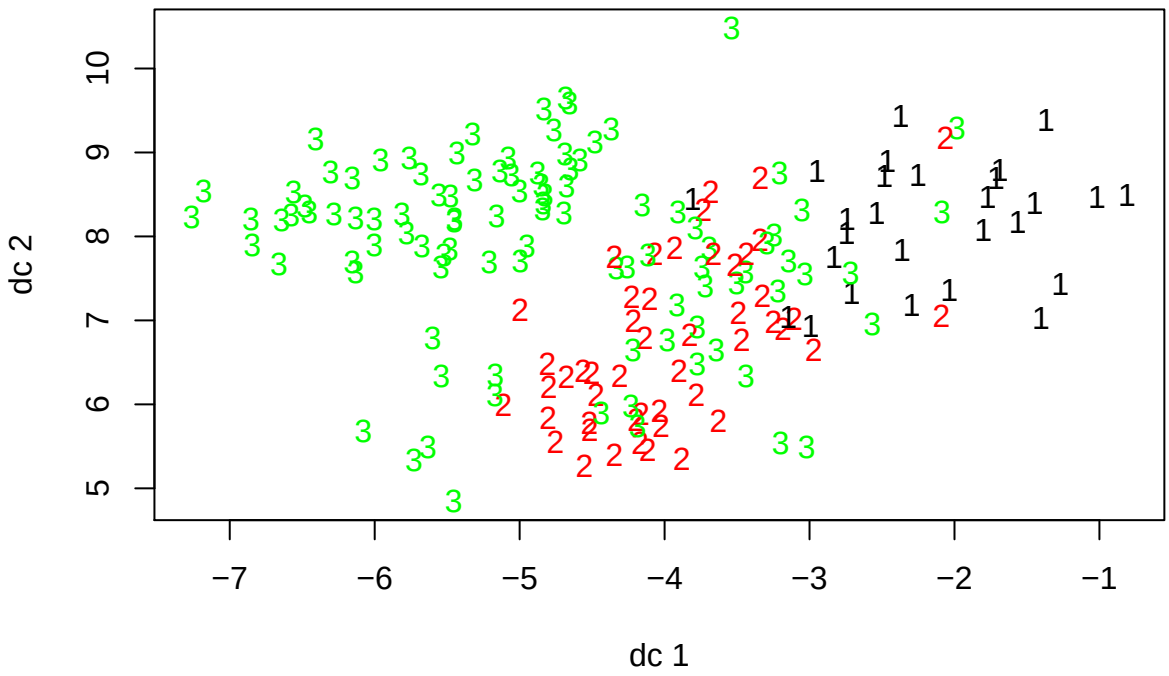
Now we shall use the function clusterByKmeans() to cluster the wine data and the iris data.

K means for the wine data without scaling:

```
# Obtain the wine data from the package "rattle"
data(wine, package="rattle")
head(wine)
```

```
##   Type Alcohol Malic  Ash Alcalinity Magnesium Phenols Flavanoids
## 1    1   14.23  1.71 2.43       15.6       127    2.80       3.06
## 2    1   13.20  1.78 2.14       11.2       100    2.65       2.76
## 3    1   13.16  2.36 2.67       18.6       101    2.80       3.24
## 4    1   14.37  1.95 2.50       16.8       113    3.85       3.49
## 5    1   13.24  2.59 2.87       21.0       118    2.80       2.69
## 6    1   14.20  1.76 2.45       15.2       112    3.27       3.39
##   Nonflavanoids Proanthocyanins Color  Hue Dilution Proline
## 1          0.28            2.29  5.64 1.04     3.92    1065
## 2          0.26            1.28  4.38 1.05     3.40    1050
## 3          0.30            2.81  5.68 1.03     3.17    1185
## 4          0.24            2.18  7.80 0.86     3.45    1480
## 5          0.39            1.82  4.32 1.04     2.93     735
## 6          0.34            1.97  6.75 1.05     2.85    1450
```

```
# Exclude the "Type" variable from the data inputs
# Assign the others to data.train
data.train <- wine[-1]
# Use set seed() so that the clustering results can be reproducible
set.seed(37810)
# Use k-means method to cluster the wines into 3 groups
fit.km <- clusterByKmeans(data.train,3)
# Visualize the clustering results using plotcluster() from the fpc library
plotcluster(data.train, fit.km)
```



As can be seen from the above plot, there exist lots of overlaps between clusters, for instance, between cluster 2 and cluster 3. It is not suitable to say that the data are well-separated.

We can compare the clustering results by k-means and the original classification of the 178 data indicated by the variable "Type":

```
# A comparison of the clustering results by k-means
# and the original classification indicated by "Type"
wt.km <- table(wine$Type, fit.km)
wt.km
```

```
##    fit.km
##      1  2  3
##   1  0  1 58
##   2 25 20 26
##   3  2 28 18
```

```
# Calculate the adjusted rand index, which is used for
# quantifying to what extent the two ways of clustering agree with each other.
randIndex(wt.km)
```

```
##       ARI
## 0.1984624
```

The above table shows directly that differences between the two indeed exist. The value of the adjusted rand index is 0.1984624, which is not quite close to 1. Note that the adjusted rand index which is a measure of the

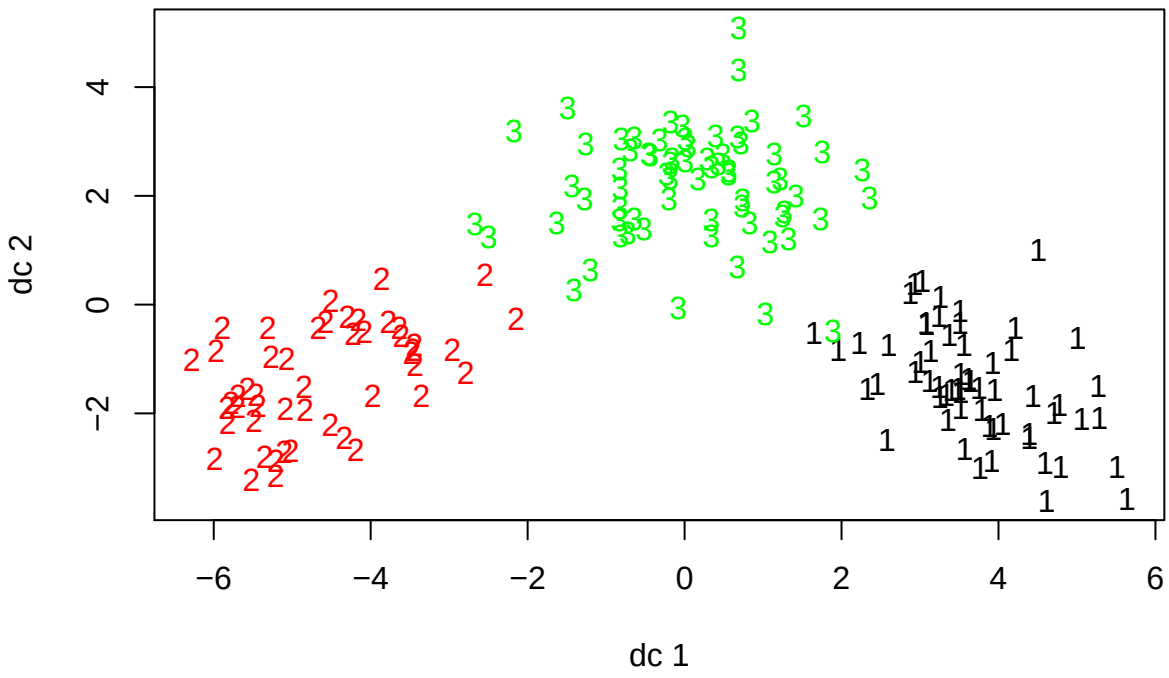
degree of agreement between two partitions takes values between -1 (no agreement) and 1 (perfect agreement). Hence, the partition obtained by K means does not match the original types of the wine very well.

Now we repeat the above exercise using scaled data:

```
# Obtain the wine data from the package "rattle"
data(wine, package="rattle")
head(wine)
```

```
##   Type Alcohol Malic  Ash Alcalinity Magnesium Phenols Flavanoids
## 1    1   14.23  1.71 2.43       15.6       127    2.80       3.06
## 2    1   13.20  1.78 2.14       11.2       100    2.65       2.76
## 3    1   13.16  2.36 2.67       18.6       101    2.80       3.24
## 4    1   14.37  1.95 2.50       16.8       113    3.85       3.49
## 5    1   13.24  2.59 2.87       21.0       118    2.80       2.69
## 6    1   14.20  1.76 2.45       15.2       112    3.27       3.39
##   Nonflavanoids Proanthocyanins Color  Hue Dilution Proline
## 1          0.28            2.29  5.64 1.04     3.92    1065
## 2          0.26            1.28  4.38 1.05     3.40    1050
## 3          0.30            2.81  5.68 1.03     3.17    1185
## 4          0.24            2.18  7.80 0.86     3.45    1480
## 5          0.39            1.82  4.32 1.04     2.93     735
## 6          0.34            1.97  6.75 1.05     2.85    1450
```

```
# Exclude the "Type" variable from the data inputs,
# center and scale the rest of the data and assign the rest to data.train
data.train <- scale(wine[-1])
# Use set seed() so that the clustering results can be reproducible.
# set.seed(37810)
# Use k-means method to cluster the wines into 3 groups
fit.km <- clusterByKmeans(data.train,3)
# Visualize the clustering results using plotcluster() from the fpc library
plotcluster(data.train, fit.km)
```



The command, data.train <- scale(wine[-1]), is used for center and scale the wine data excluding the variable Type. More specifically, for each column of wine[-1], first substract the corresponding column mean and then dividing that centered column by its standard deviation. In data.train, each column has a mean of zero.

Using the scaled data, we obtain a better clustering result in the sense that now the clusters seem to be well-separated, despite a minor overlap between cluster 1 and cluster 3.

We can compare the clustering results by k-means and the original classification of the 178 data indicated by the variable "Type":

```
# A comparison of the clustering results by k-means
# and the original claassification indicated by "Type"
wt.km <- table(wine$Type, fit.km)
wt.km
```

```
##    fit.km
##      1  2  3
##   1 57  0  2
##   2  3  1 67
##   3  0 48  0
```

```
randIndex(wt.km)
```

```
##       ARI
## 0.8951549
```

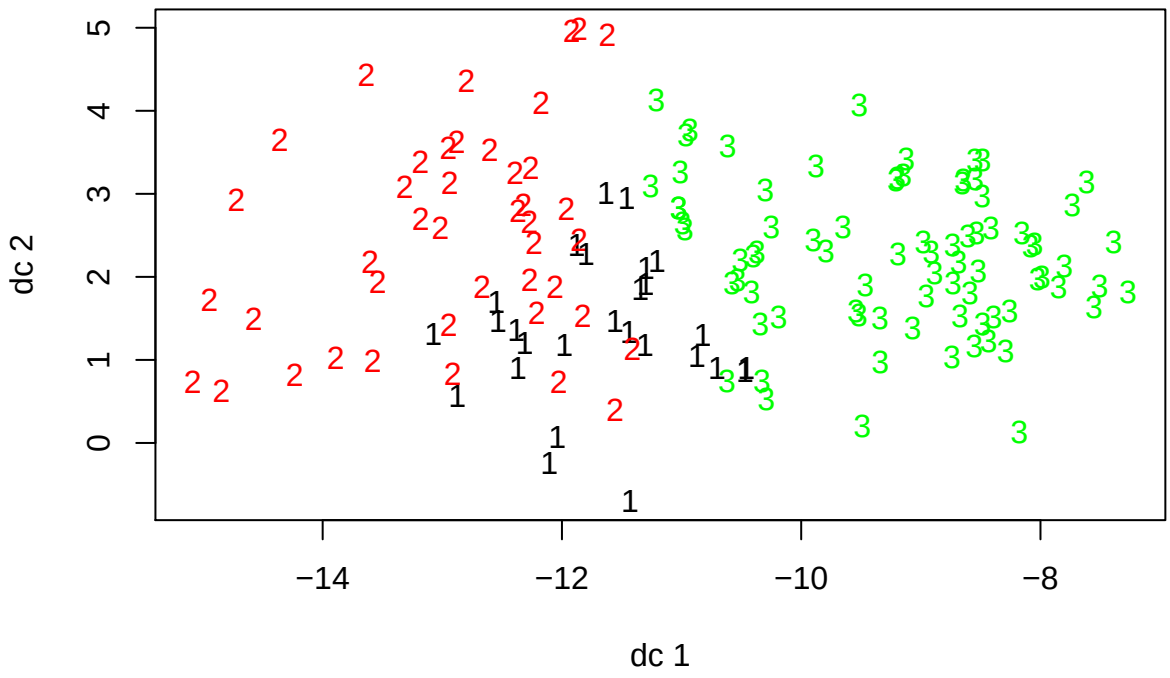
Now the table shows that there are fewer missing samples than in the unscaled case. The value of the adjusted rand index is 0.6873931, which is much larger than in the unscaled case. Hence, scaling helps improve the performance of K means in the sense that after scaling the clustering result using K means is more similar to the original classification.

In the following we show repeat the above exercise for the iris dataset. We first use the original dataset which is not scaled.

```
# Obtain the iris data
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# Exclude the "Species" variable and assign the rest to data.train
data.train <- iris[-5]
# Use set seed() so that the clustering results can be reproducible
set.seed(37810)
# Use k-means method to cluster the data into 3 groups
fit.km <- clusterByKmeans(data.train,3)
# Visualize the clustering results using plotcluster() from the fpc library
plotcluster(data.train, fit.km)
```



The plot here shows some overlaps, in particular between cluster 1 and cluster 2 and between cluster 1 and cluster 3, as a result of which k-means does not work very well for clustering this dataset.

We can compare the clustering results by k-means and the original classification of the data indicated by the variable "species":

```
fs.km <- table(iris$Species,fit.km)
fs.km
```

```
##             fit.km
##               1  2  3
##   setosa      0  0 50
##   versicolor 25  3 22
##   virginica   2 40  8
```

```
randIndex(fs.km)
```

```
##       ARI
## 0.4305639
```

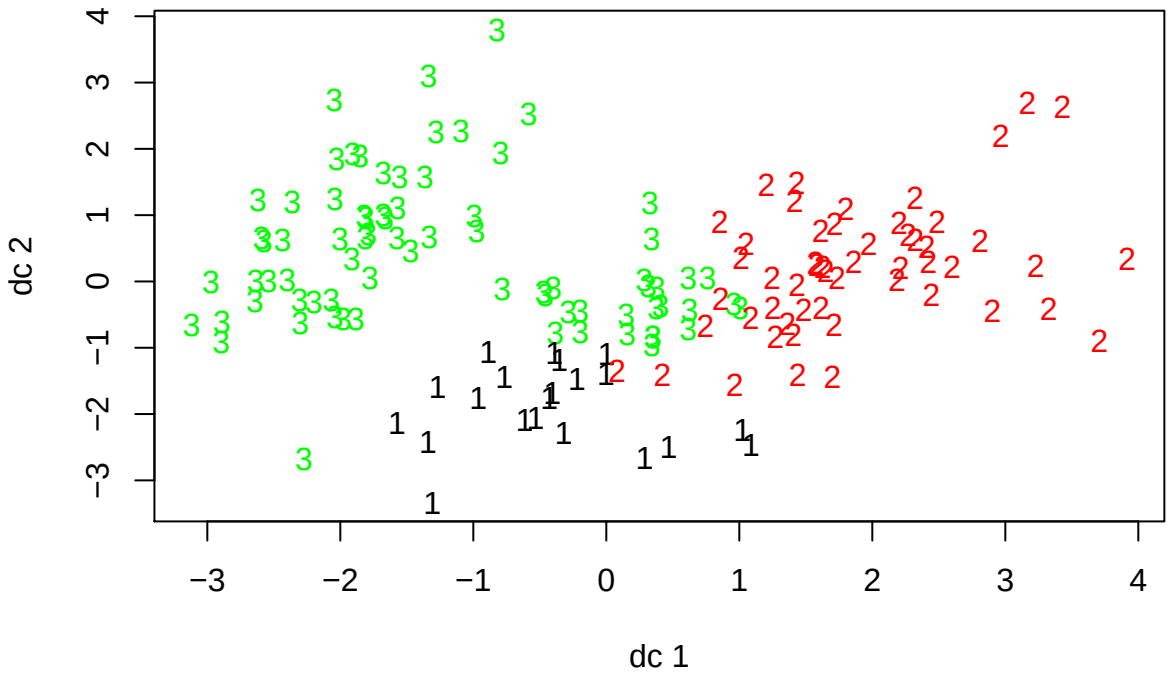
The adjusted rand index here is 0.4305639, which is not quite close to 1. So the algorithm's clusters do not match the original partition very well.

Now we instead use the scaled iris dataset:

```r
# Obtain the iris data
data(iris)
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```r
# Exclude the "Species" variable and assign the rest to data.train
data.train <- scale(iris[-5])
# Use set seed() so that the clustering results can be reproducible
set.seed(37810)
# Use k-means method to cluster the data into 3 groups
fit.km <- clusterByKmeans(data.train,3)
# Visualize the clustering results using plotcluster() from the fpc library
plotcluster(data.train, fit.km)
```



As shown in the plot, there are still some overlaps between clusters, indicating that k-means does not work very well even for this scaled data set.

To quantify the comparison, we can still apply randIndex()

```
fs.km <- table(iris$Species,fit.km)
fs.km
```

```
##             fit.km
##               1  2  3
##   setosa      0  0 50
##   versicolor 19 12 19
##   virginica   2 41  7
```

```
randIndex(fs.km)
```

```
##       ARI
## 0.3910336
```

The adjusted rand index now is 0.3910336, which is even a little bit smaller than the unscaled case. In other words, for the iris dataset, scaling does not help improving the effectiveness of the k-means.

As a summary, scaling is useful for improving the performance of K means for the wine data but not for the iris data.